**Assignment**

**CSA0805 – Python Programming**

|  |  |
| --- | --- |
| **Register Number** | **192324196** |
| **Name** | **K Rupesh Reddy** |

**Title:** **File Encryption Utility**

**Problem Statement:** Write a Python program that encrypts sensitive files using symmetric or asymmetric encryption algorithms from the cryptography module, providing options for key management and secure storage

**Code:**

**from cryptography.hazmat.primitives.asymmetric import rsa, padding**

**from cryptography.hazmat.primitives import serialization, hashes**

**from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC**

**from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes**

**from cryptography.hazmat.backends import default\_backend**

**from cryptography.hazmat.primitives import padding as sym\_padding**

**from cryptography.hazmat.primitives.keywrap import aes\_key\_wrap, aes\_key\_unwrap**

**from cryptography.hazmat.primitives.kdf.scrypt import Scrypt**

**import os**

**# Constants**

**AES\_KEY\_SIZE = 32 # 256-bit key for AES**

**RSA\_KEY\_SIZE = 2048**

**SALT\_SIZE = 16**

**def generate\_rsa\_keys():**

**"""Generate RSA public and private keys."""**

**private\_key = rsa.generate\_private\_key(**

**public\_exponent=65537,**

**key\_size=RSA\_KEY\_SIZE,**

**backend=default\_backend()**

**)**

**public\_key = private\_key.public\_key()**

**return private\_key, public\_key**

**def serialize\_rsa\_key(private\_key, password=None):**

**"""Serialize RSA private key with optional password protection."""**

**encryption\_algorithm = (**

**serialization.BestAvailableEncryption(password.encode())**

**if password else serialization.NoEncryption()**

**)**

**return private\_key.private\_bytes(**

**encoding=serialization.Encoding.PEM,**

**format=serialization.PrivateFormat.PKCS8,**

**encryption\_algorithm=encryption\_algorithm**

**)**

**def deserialize\_rsa\_key(pem, password=None):**

**"""Deserialize RSA private key with optional password."""**

**return serialization.load\_pem\_private\_key(**

**pem,**

**password=password.encode() if password else None,**

**backend=default\_backend()**

**)**

**def rsa\_encrypt(public\_key, data):**

**"""Encrypt data using RSA."""**

**return public\_key.encrypt(**

**data,**

**padding.OAEP(**

**mgf=padding.MGF1(algorithm=hashes.SHA256()),**

**algorithm=hashes.SHA256(),**

**label=None**

**)**

**)**

**def rsa\_decrypt(private\_key, encrypted\_data):**

**"""Decrypt data using RSA."""**

**return private\_key.decrypt(**

**encrypted\_data,**

**padding.OAEP(**

**mgf=padding.MGF1(algorithm=hashes.SHA256()),**

**algorithm=hashes.SHA256(),**

**label=None**

**)**

**)**

**def generate\_aes\_key():**

**"""Generate a random AES key."""**

**return os.urandom(AES\_KEY\_SIZE)**

**def aes\_encrypt(key, data):**

**"""Encrypt data using AES."""**

**iv = os.urandom(16) # AES block size is 16 bytes**

**cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default\_backend())**

**encryptor = cipher.encryptor()**

**# Padding the data to be a multiple of the block size**

**padder = sym\_padding.PKCS7(algorithms.AES.block\_size).padder()**

**padded\_data = padder.update(data) + padder.finalize()**

**return iv + encryptor.update(padded\_data) + encryptor.finalize()**

**def aes\_decrypt(key, encrypted\_data):**

**"""Decrypt data using AES."""**

**iv = encrypted\_data[:16]**

**cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default\_backend())**

**decryptor = cipher.decryptor()**

**decrypted\_padded\_data = decryptor.update(encrypted\_data[16:]) + decryptor.finalize()**

**# Remove padding**

**unpadder = sym\_padding.PKCS7(algorithms.AES.block\_size).unpadder()**

**return unpadder.update(decrypted\_padded\_data) + unpadder.finalize()**

**def encrypt\_file(filepath, public\_key):**

**"""Encrypt a file using AES and encrypt the AES key with RSA."""**

**# Read the file data**

**with open(filepath, 'rb') as file:**

**file\_data = file.read()**

**# Generate AES key**

**aes\_key = generate\_aes\_key()**

**# Encrypt file data with AES**

**encrypted\_data = aes\_encrypt(aes\_key, file\_data)**

**# Encrypt AES key with RSA**

**encrypted\_aes\_key = rsa\_encrypt(public\_key, aes\_key)**

**# Write encrypted AES key + encrypted file data to file**

**with open(filepath + ".enc", 'wb') as file:**

**file.write(encrypted\_aes\_key + encrypted\_data)**

**print(f"File encrypted successfully: {filepath}.enc")**

**def decrypt\_file(encrypted\_filepath, private\_key, password=None):**

**"""Decrypt an encrypted file using the private RSA key."""**

**with open(encrypted\_filepath, 'rb') as file:**

**encrypted\_data = file.read()**

**# Extract the encrypted AES key and encrypted file data**

**encrypted\_aes\_key = encrypted\_data[:RSA\_KEY\_SIZE // 8]**

**encrypted\_file\_data = encrypted\_data[RSA\_KEY\_SIZE // 8:]**

**# Decrypt AES key with RSA**

**aes\_key = rsa\_decrypt(private\_key, encrypted\_aes\_key)**

**# Decrypt file data with AES**

**file\_data = aes\_decrypt(aes\_key, encrypted\_file\_data)**

**# Write decrypted data to a new file**

**output\_filepath = encrypted\_filepath.replace(".enc", ".dec")**

**with open(output\_filepath, 'wb') as file:**

**file.write(file\_data)**

**print(f"File decrypted successfully: {output\_filepath}")**

**# Example usage:**

**if \_\_name\_\_ == "\_\_main\_\_":**

**# Generate RSA keys**

**private\_key, public\_key = generate\_rsa\_keys()**

**# Serialize private key to save it securely**

**password = "my\_secret\_password"**

**serialized\_private\_key = serialize\_rsa\_key(private\_key, password)**

**# Save the private key to a file (make sure this is stored securely!)**

**with open("private\_key.pem", "wb") as key\_file:**

**key\_file.write(serialized\_private\_key)**

**# Encrypt a file**

**filepath\_to\_encrypt = "sensitive\_data.txt"**

**encrypt\_file(filepath\_to\_encrypt, public\_key)**

**# Load the private key back from the file**

**with open("private\_key.pem", "rb") as key\_file:**

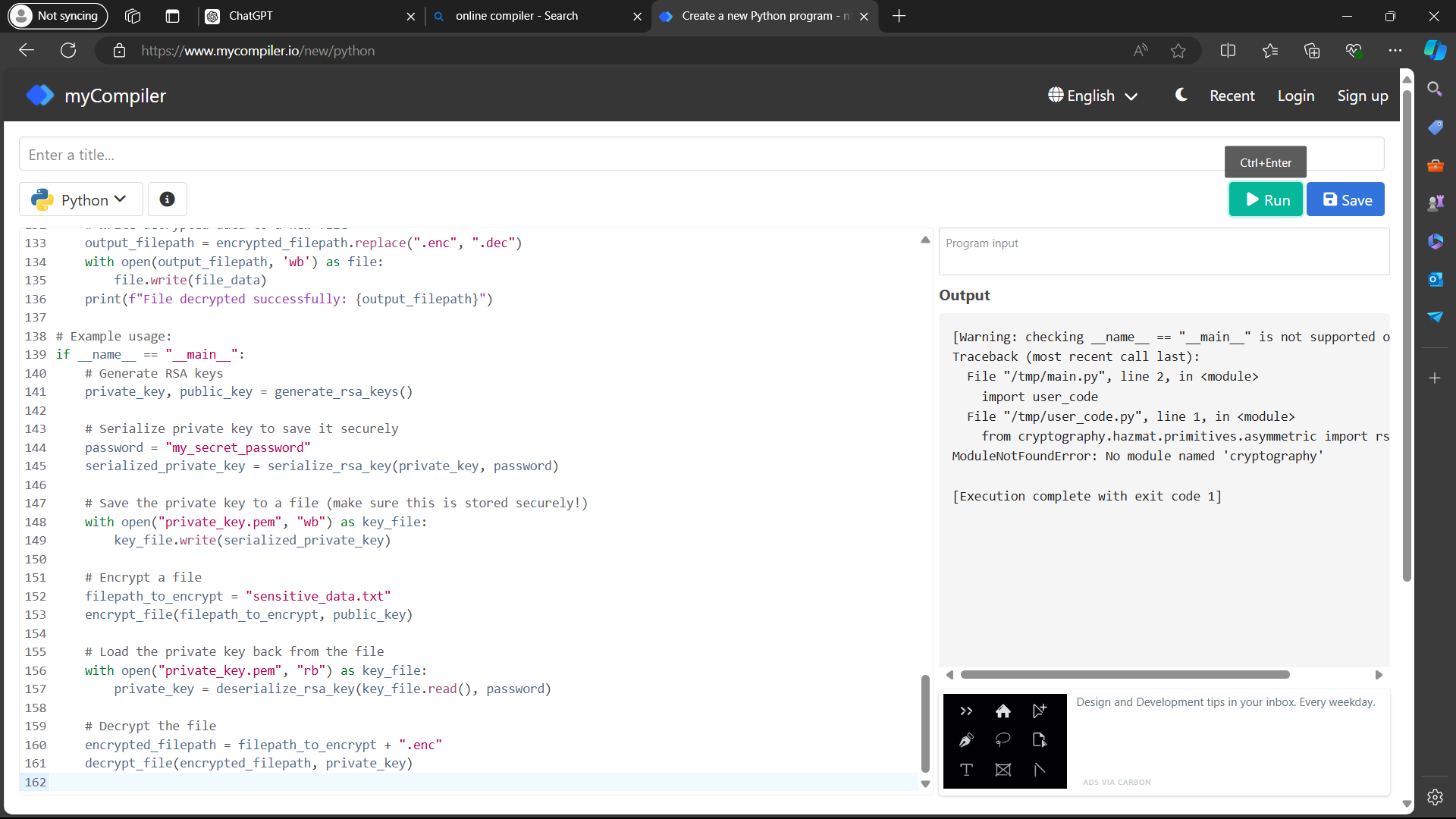
**private\_key = deserialize\_rsa\_key(key\_file.read(), password)**

**# Decrypt the file**

**encrypted\_filepath = filepath\_to\_encrypt + ".enc"**

**decrypt\_file(encrypted\_filepath, private\_key)**

**Output Screen Shots**

****

**Conclusion:**

**This Python program demonstrates a practical approach to file encryption and decryption using symmetric encryption with AES, leveraging the cryptography library for secure operations. Here’s a summary of the key points**